Table of Contents

DavinciL1PProgramMemory

Using Davinci's L1P region as non-cached program memory

Problem

On the Davinci, the L1P memory region can be mapped as either cache, RAM or a combination of both. After restart, the L1P configuration register (L1PCFG) is configured to all-cache mode by default. In a Codec Engine application, DSP/BIOS can be used to configure part or all of L1P into RAM. Here's an example of how to configure all of L1P into program memory:

```
prog.module("GBL").C64PLUSL1PCFG      = "0k";

var L1PSRAM = bios.MEM.create("L1PSRAM");
L1PSRAM.len = 0x7000;
L1PSRAM.base = 0x11E08000;
L1PSRAM.createHeap = false;
L1PSRAM.space = "code";
```

DSP/BIOS modifies the configuration register in its boot code. However, by the time the instruction is executed, some code would already have been cached in the L1P area. Hence it is not advised to load code or initialized data into the L1P area as it has a chance to be overwritten before the cache mode is set.

This topic discusses how to leverage as program memory the RAM that becomes available when L1P is not configured as 100% cache.

Solution

To simplify things, the solution proposed makes use of copy tables (see http://focus.ti.com/lit/an/spraa46/spraa46.pdf for details) to bring code into L1P. Assuming we want to bring some code myCode.obj into L1P, we can load the code into DDR and specify L1P as a run address in the linker cmd file:

```
.func1 { myCode.obj(.text) } load = DDR, run = L1PSRAM, table(_func1_copy_table)
.ovly > DDR
```

Then in the user code (e.g. main()), one can do the following

```
extern void EDMACOPY_copy_in(COPY_TABLE *tp);
extern COPY_TABLE func1_copy_table;


Void main (Int argc, Char * argv [])
{
    // Bring the code from external memory to L1P
    EDMACOPY_copy_in(&func1_copy_table);

   ....
}
```

The 'EDMACOPY_copy_in()' function shown here is used as an alternative to the copy_in() function provided by the RTS library in the code generation tools to bring the code section from external memory to L1P. This function uses the EDMA to perform the transfer instead of memcpy. This is necessary because the

L1P memory region cannot be written to by the CPU. An example implementation of EDMACOPY_copy_in() is currently available upon request. Although anyone familiar with the EDMA should be able to easily write this function. Taking a peek at the existing implementation for copy_in() in rts.src in the code generation tools should help you figure out how to work with the copy table structure.

-- VincentWan - 15 Nov 2006